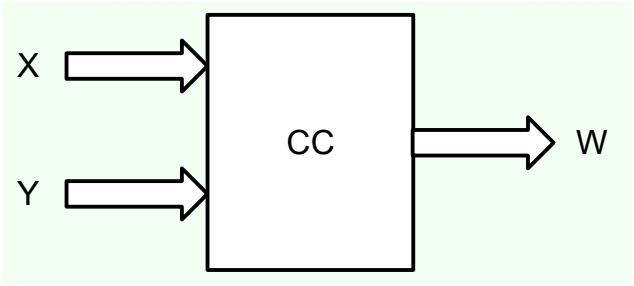
ESC201T : Introduction to Electronics

Lecture 35: Combination circuit design-2

B. Mazhari
Dept. of EE, IIT Kanpur

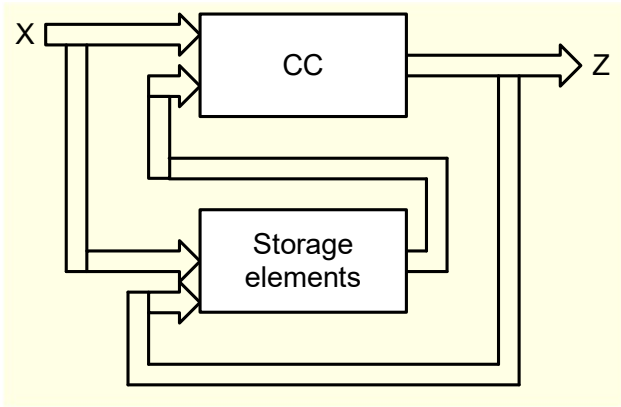
Digital Circuits

Combinational Circuits



Output is determined by current values of inputs only.

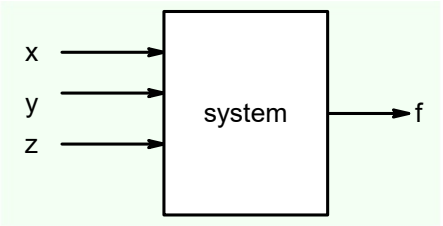
Sequential Circuits



Output is determined in general by current values of inputs and past values of inputs/outputs as well.

Design Flow

System Description



Truth Table

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

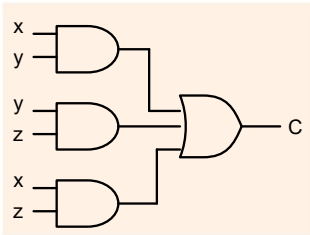
Boolean Expression

$$f = \bar{x}.\bar{y}.z + \bar{x}.y.z + x.\bar{y}.z + x.y.z$$

Minimized Boolean Expression

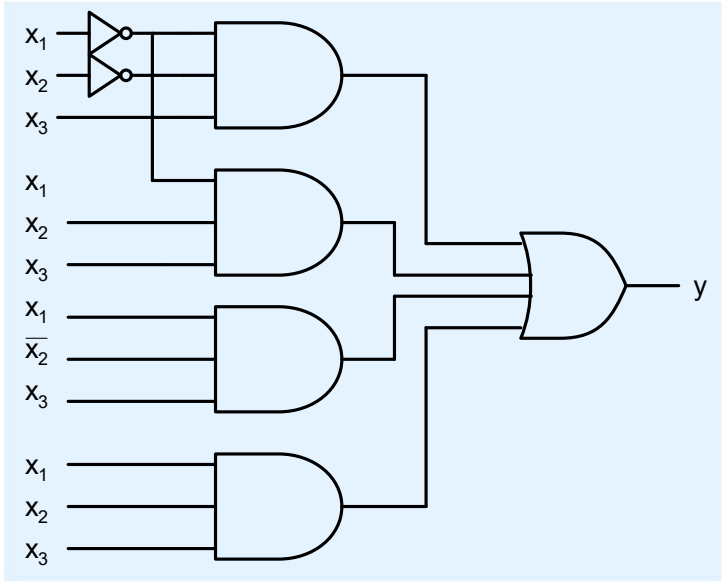
$$\Rightarrow f = \bar{x}.\bar{z} + x.z$$

Gate Netlist



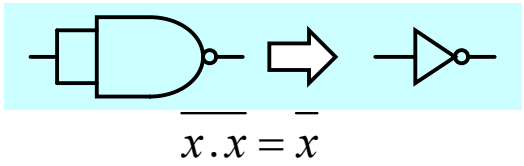
Mapping of Boolean expression to a Network of gates available in the library

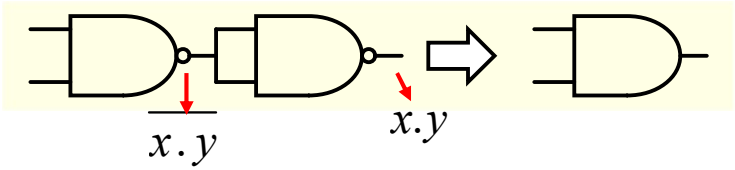
$$y = \overline{x_1} \cdot \overline{x_2} \cdot x_3 + \overline{x_1} \cdot x_2 \cdot x_3 + x_1 \cdot \overline{x_2} \cdot x_3 + x_1 \cdot x_2 \cdot x_3$$

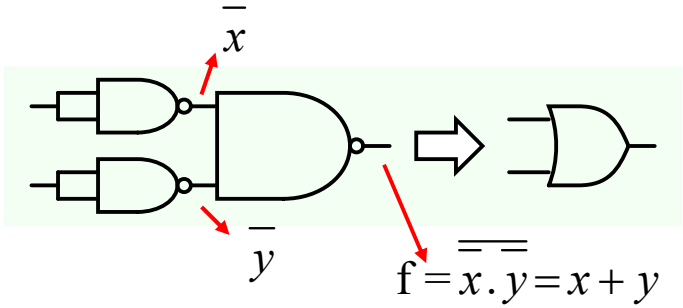


Library of available Gates	Cost
Inverter	1
Two input NAND	2
Three input NAND	3
AND-OR-Invert $Y = \overline{AB + C}$	3

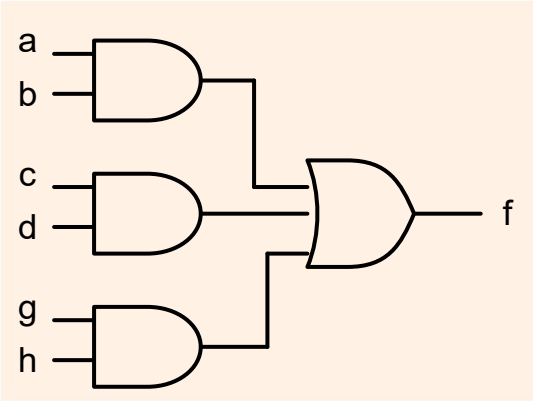
Implementation using only NAND gates

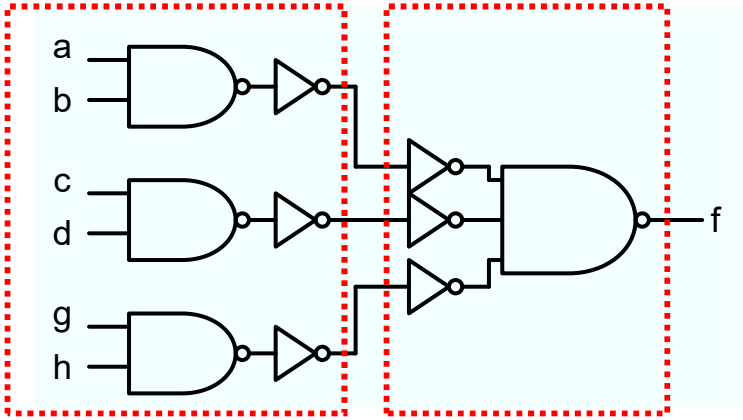


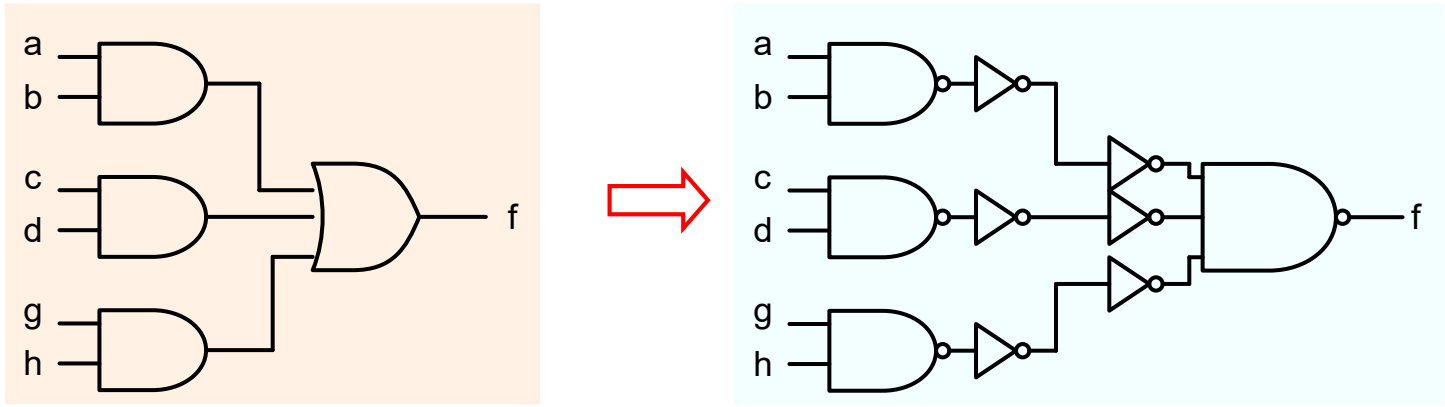


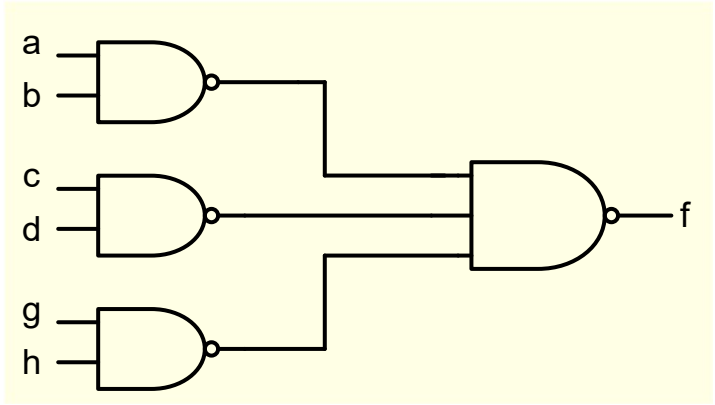


A SoP expression is easily implemented with NAND gates. $f = a.b + c.d + g.h$





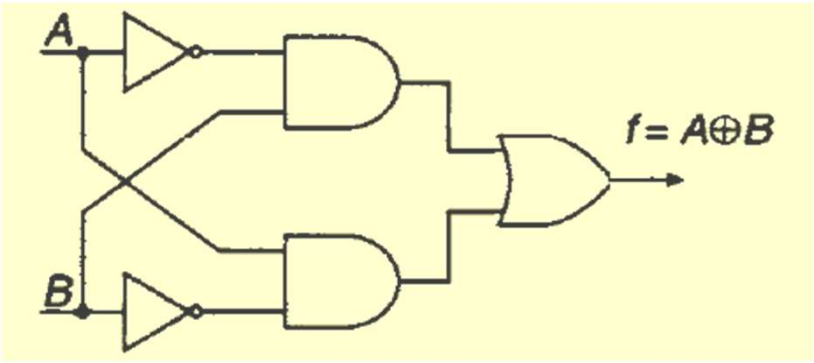


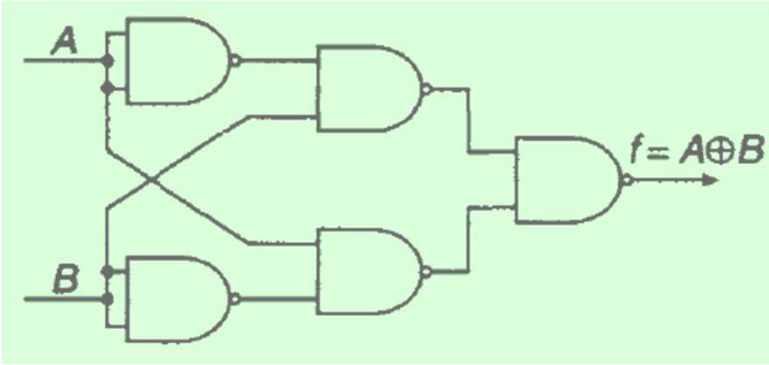


There is a one-to-one mapping between AND-OR network and NAND network

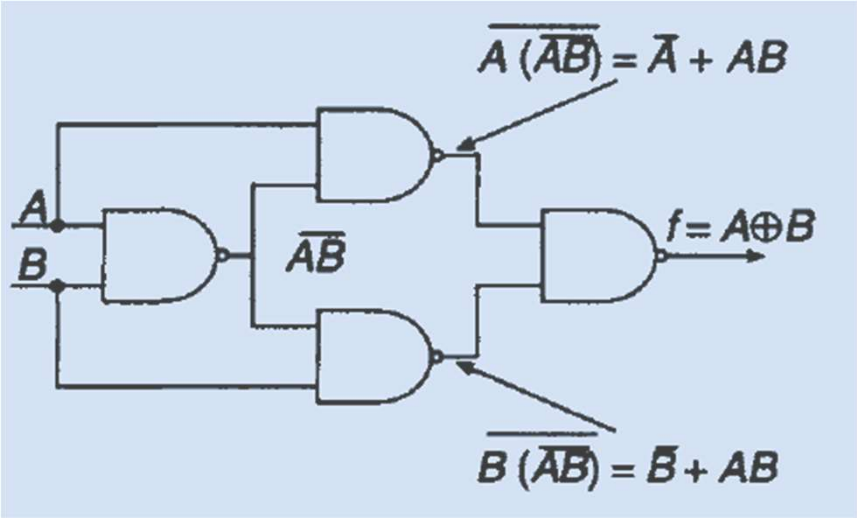
Often there is lot of further optimization that can be done

Consider implementation of XOR gate $f = \overline{A}.B + A.\overline{B}$

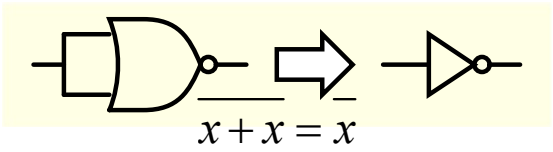


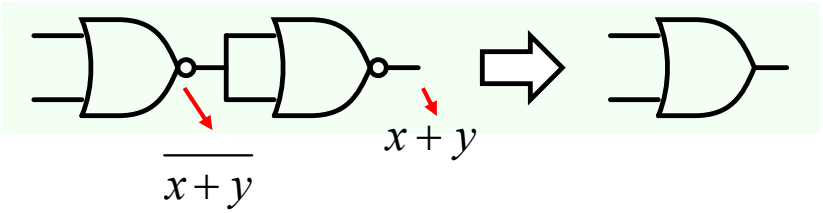


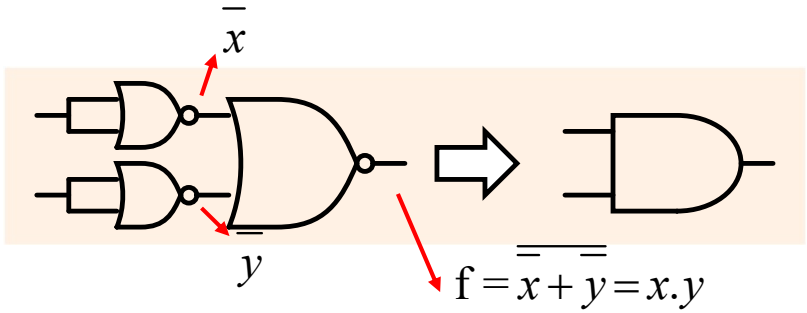
$$f = \overline{A}.B + B.\overline{B} + A.\overline{B} + A.\overline{A}$$
$$= B(\overline{A} + \overline{B}) + A(\overline{A} + \overline{B})$$



Implementation using only NOR gates



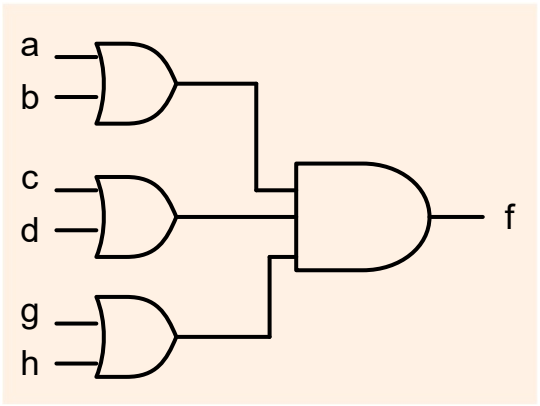


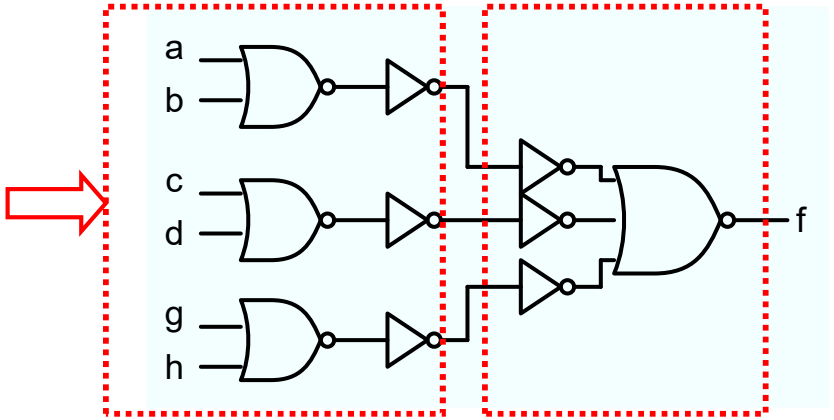


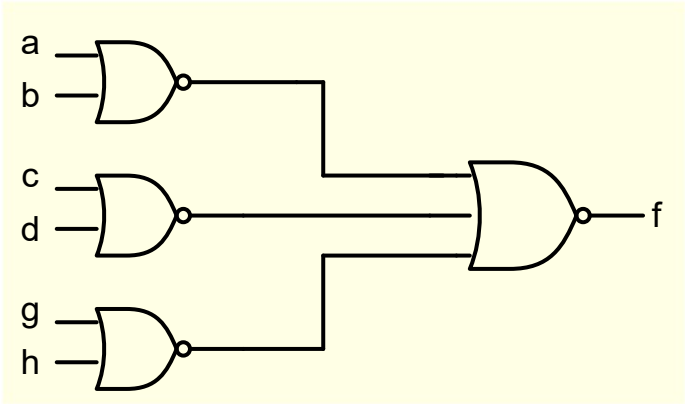
To implement using NOR gates, it is easiest to start with minimized Boolean expression in POS form

$$f=(a+b).(c+d).(g+h)$$

$$f = (a + b).(c + d).(g + h)$$







There is a one-to-one mapping between OR-AND network and NOR network

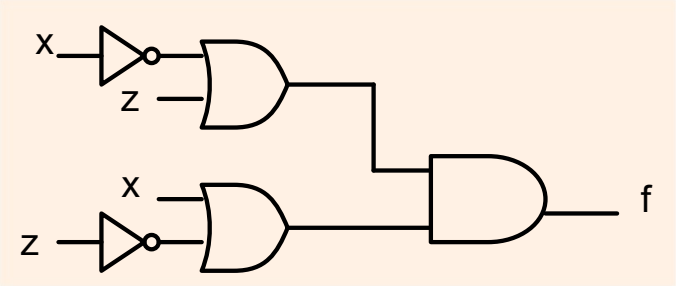
To implement SoP expression using NOR gates, determine first the corresponding PoS expression and then follow the procedure outlined earlier

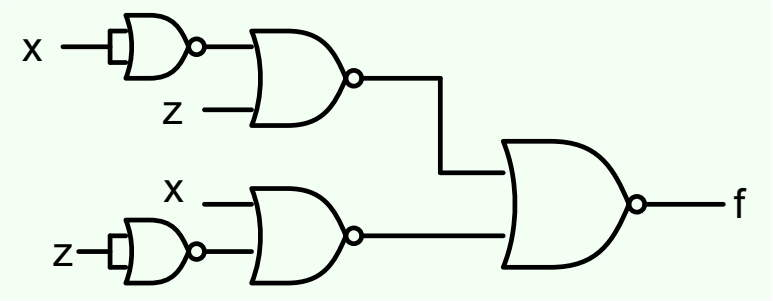
Implement $f(x,y,z) = \bar{x}\,.\bar{z} + x\,.z$ using NOR gates

$\bar{x}.\bar{z}.(y + \bar{y}) + x.z.(y + \bar{y})$

x \ yz	00	01	11	10
0	1	0	0	1
1	0	1	1	0

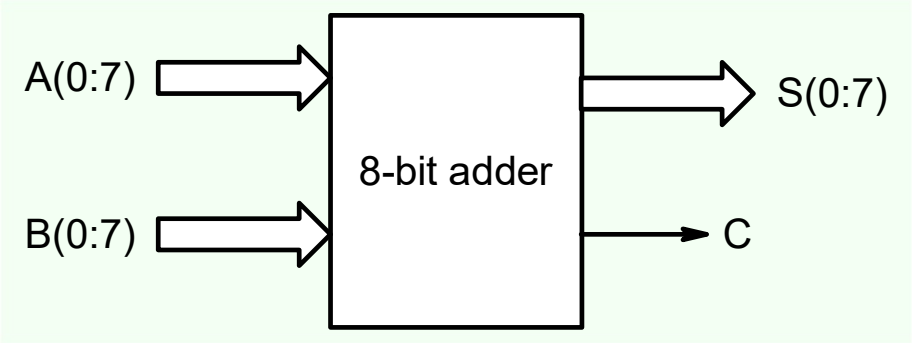
$\Rightarrow f = (\bar{x}\,. + z).(x + \bar{z})$





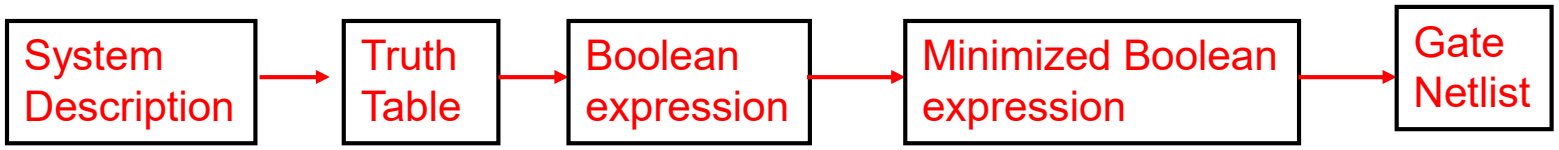
Similarly PoS expression can be implemented as NAND network by first converting it to SoP expression and then following the procedure outlined earlier

Design of Complex Combinational circuits



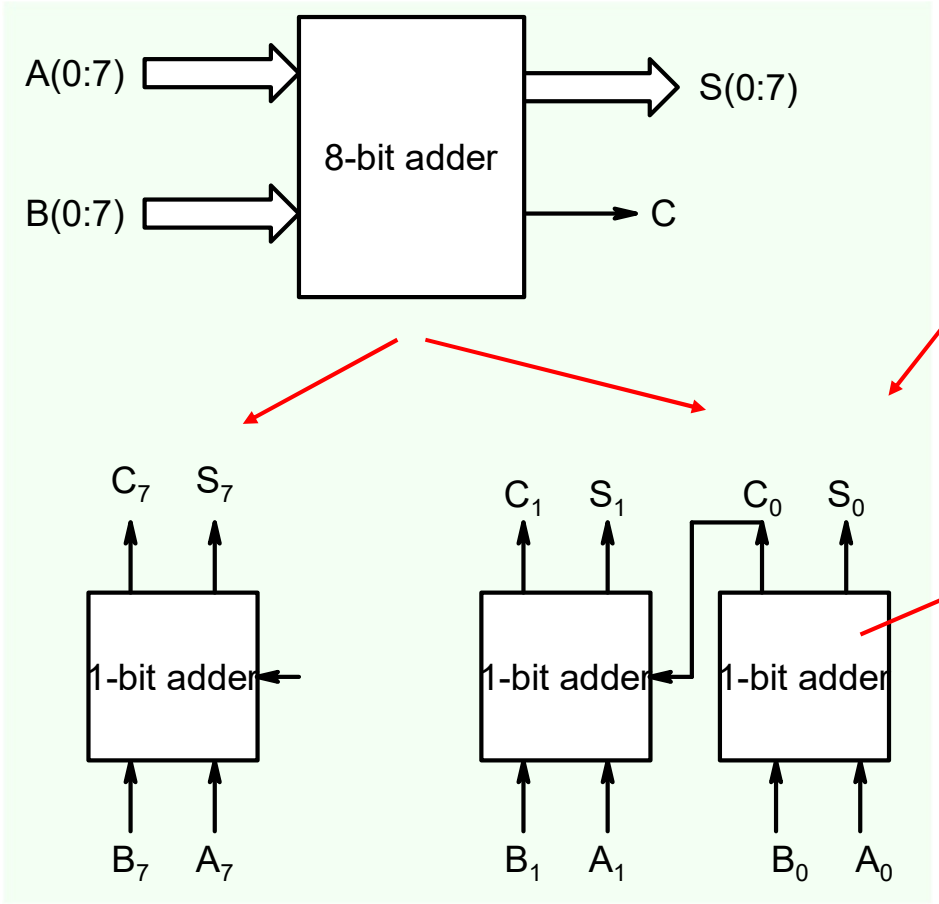
$A_7A_6...A_0$	$B_7B_6...B_0$	$S_7S_6...S_0$	C
00...0	00...0	00...0	0
00...1	00...0	00...1	0
00...0	00...1	00...1	0
⋮	⋮	⋮	⋮

Truth table has 2^{16} entries



This design approach becomes difficult to use

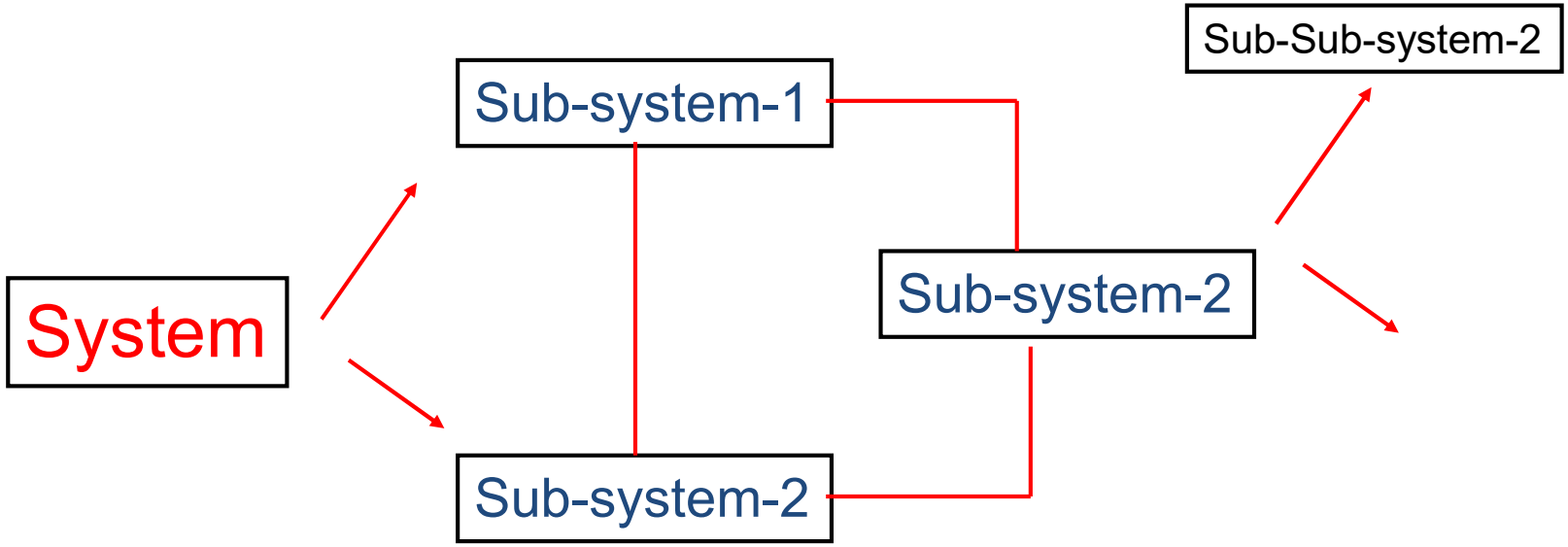
Design system as a network of sub-systems that are of manageable size and can be implemented using the earlier approach of truth table, minimization etc.




```
  1 1 0 1
+ 1 1 1 0
-------
1 1 0 1 1
```

a	b	c	S	CY
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

General Approach

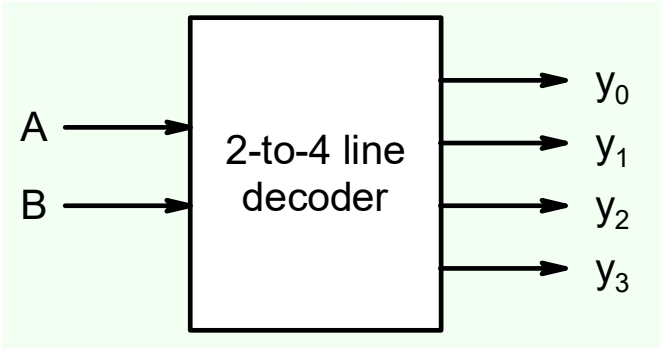


There are certain sub-systems or blocks that are used quite often such as :

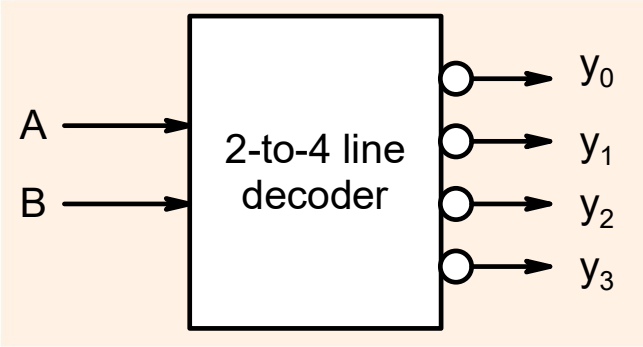
1. **decoders, encoders**
2. **Multiplexers**
3. **Adder/Subtractors, Multipliers**
4. **Comparators**
5. **Parity Generators**
6. **……..**

Decoders

Maps a smaller number of inputs to a larger set of outputs in general



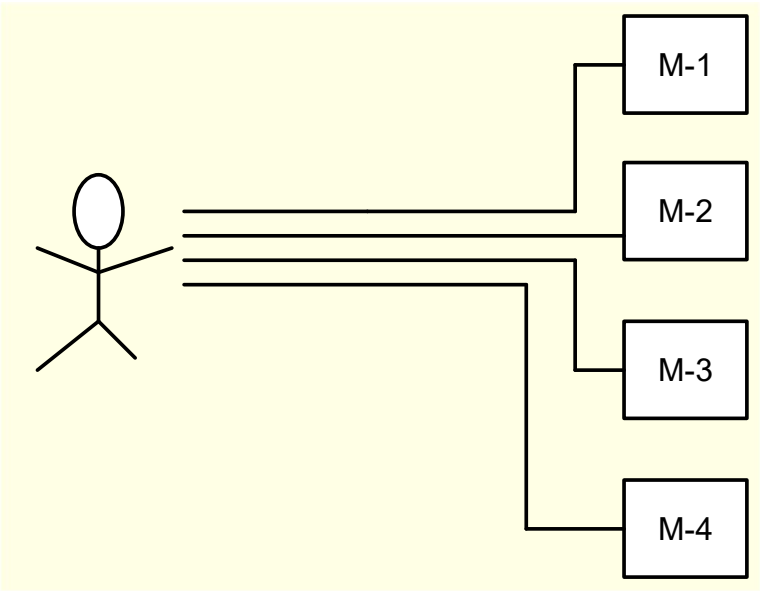
B	A	Y_0	Y_1	Y_2	Y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

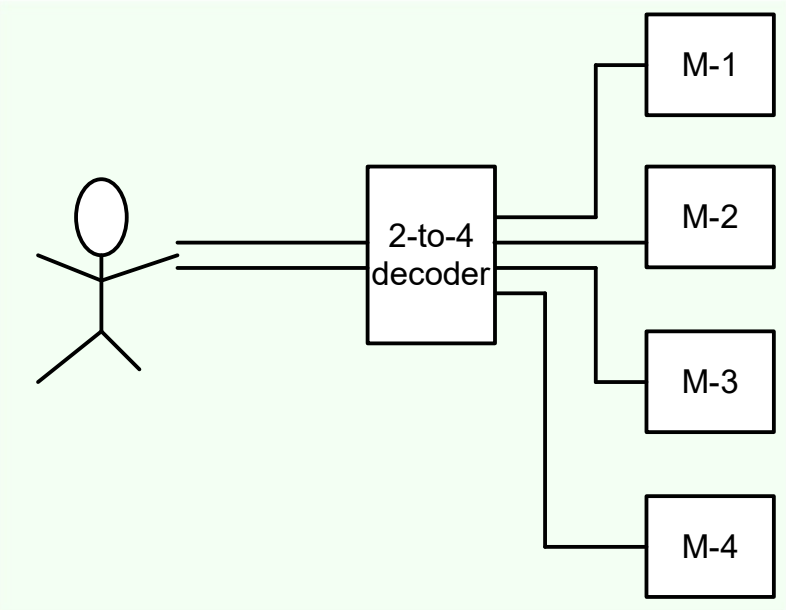


Active Low

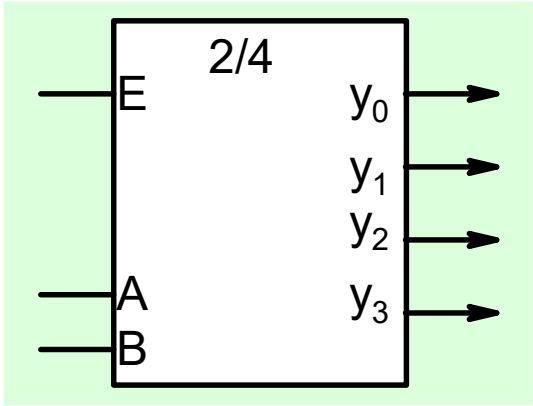
b	a	Y_0	Y_1	Y_2	Y_3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

Example

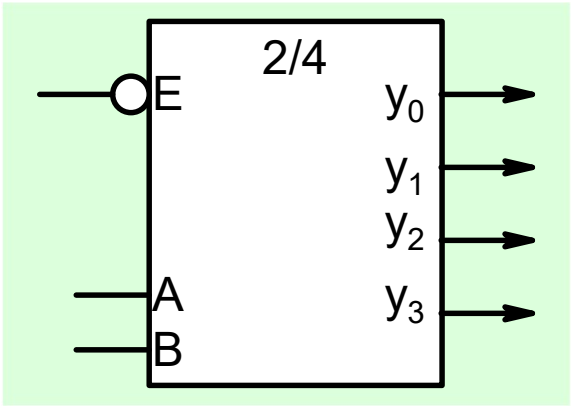




Decoder with Enable Input

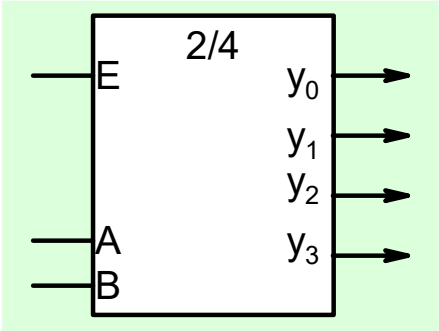


E	B	A	Y_0	Y_1	Y_2	Y_3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1



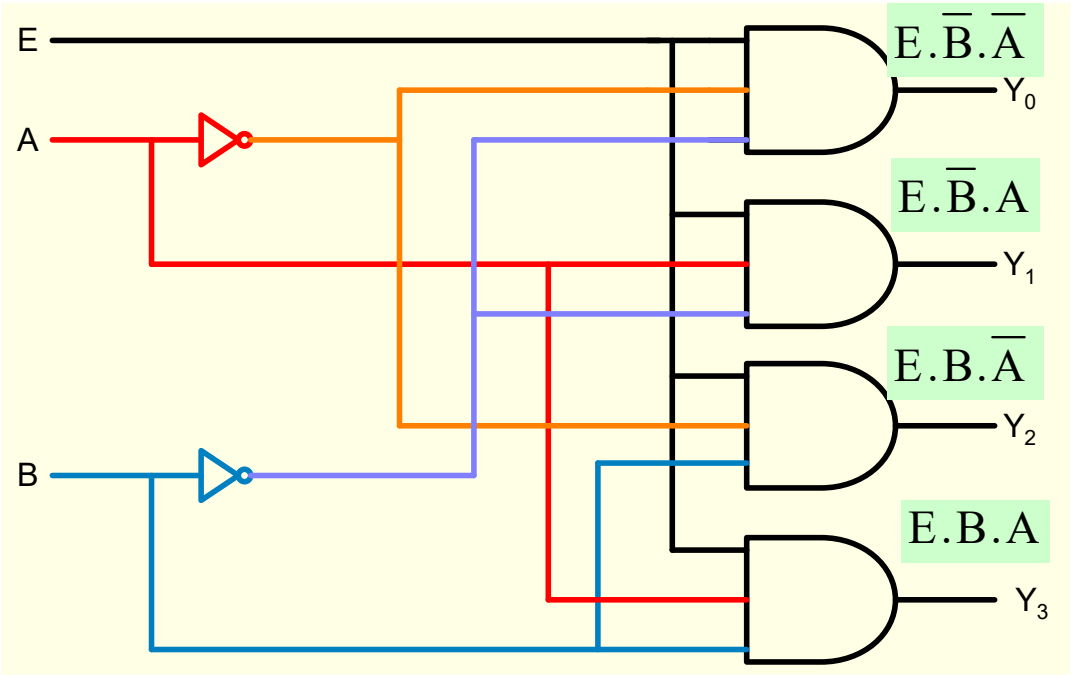
E	B	A	Y_0	Y_1	Y_2	Y_3
1	x	x	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1

Decoder: gate Implementation



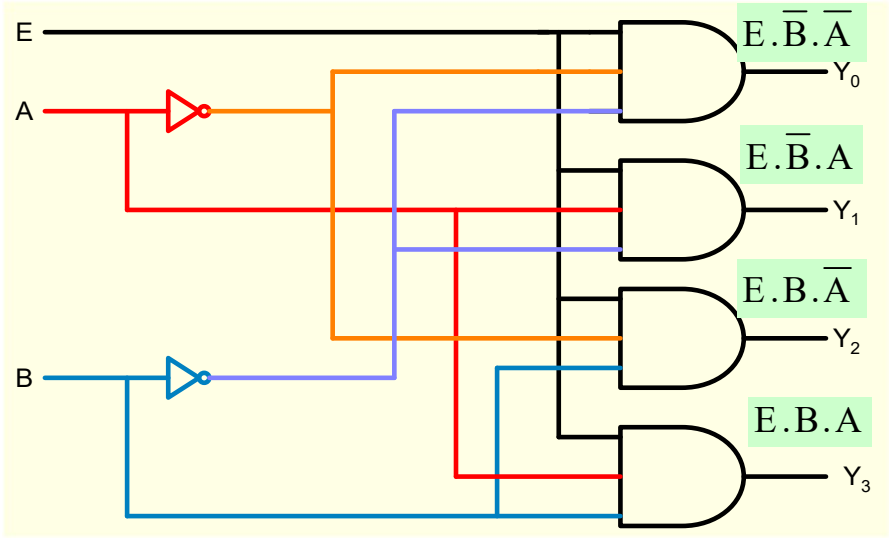
E	B	A	Y_0	Y_1	Y_2	Y_3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

$$Y_0 = E.\overline{B}.\overline{A}\ ; Y_1 = E.\overline{B}.A ; Y_2 = E.B.\overline{A}\ ; Y_3 = E.B.A$$



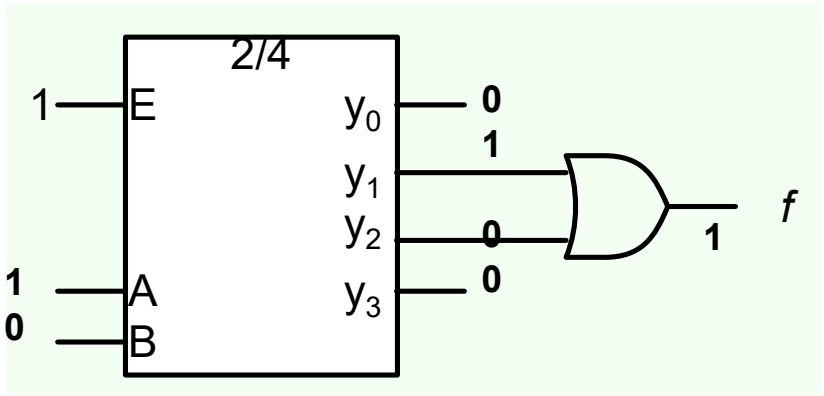
A n to 2^n decoder is a minterm generator

x	y	min term
0	0	$\overline{x}.\overline{y}$ m0
0	1	$\overline{x}.y$ m1
1	0	$x.\overline{y}$ m2
1	1	$x.y$ m3



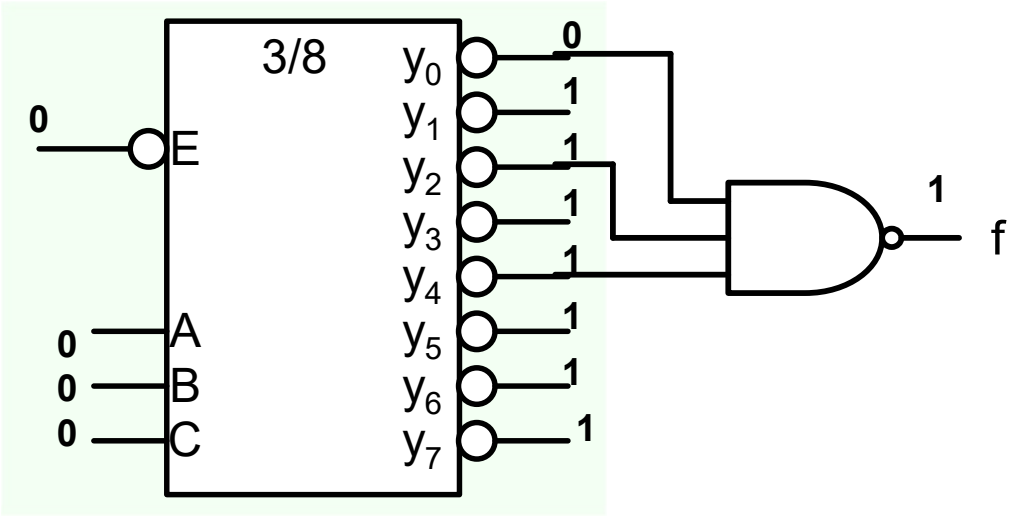
It can be used to implement any combinational circuit

B	A	f_1
0	0	0
0	1	1
1	0	1
1	1	0



Implementation of a 3-variable function with a 3-to-8 decoder

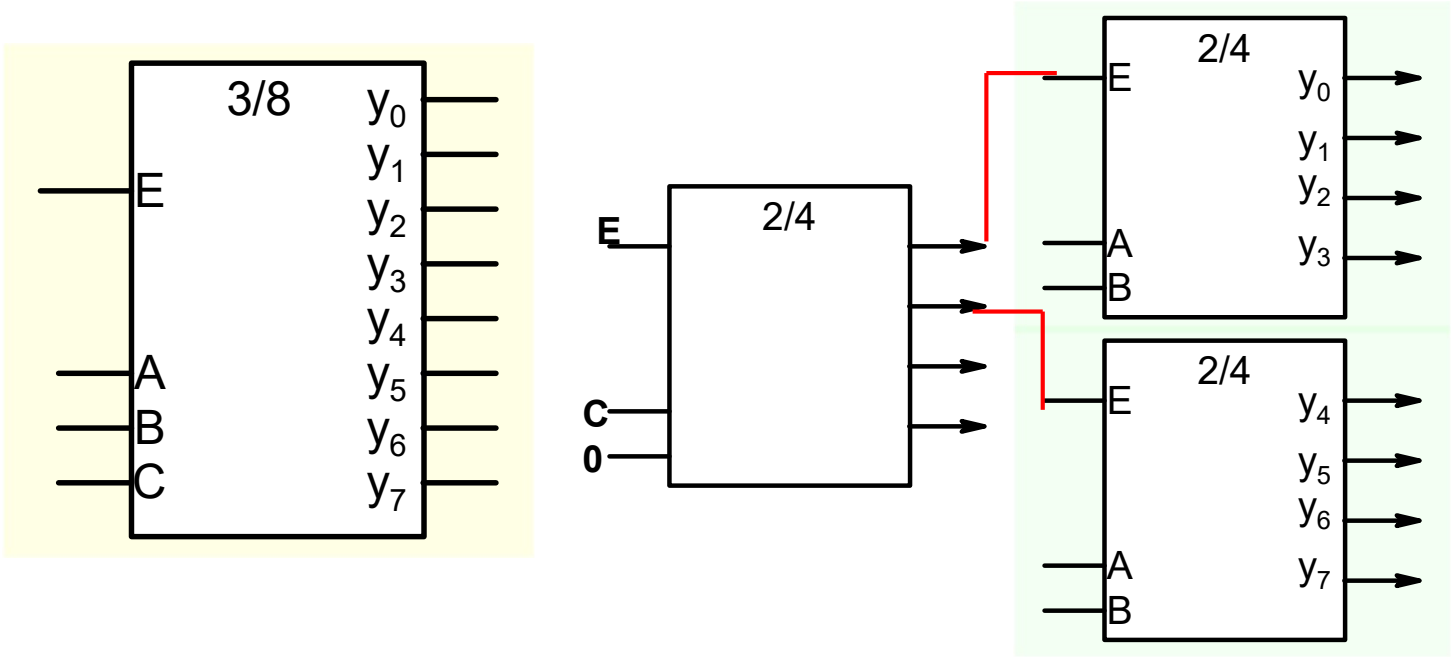
C	B	A	*f*
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0



Although it is easy to implement any combinational circuit with this method , it is often very inefficient in terms of gate utilization. Note that this method does not require any minimization.

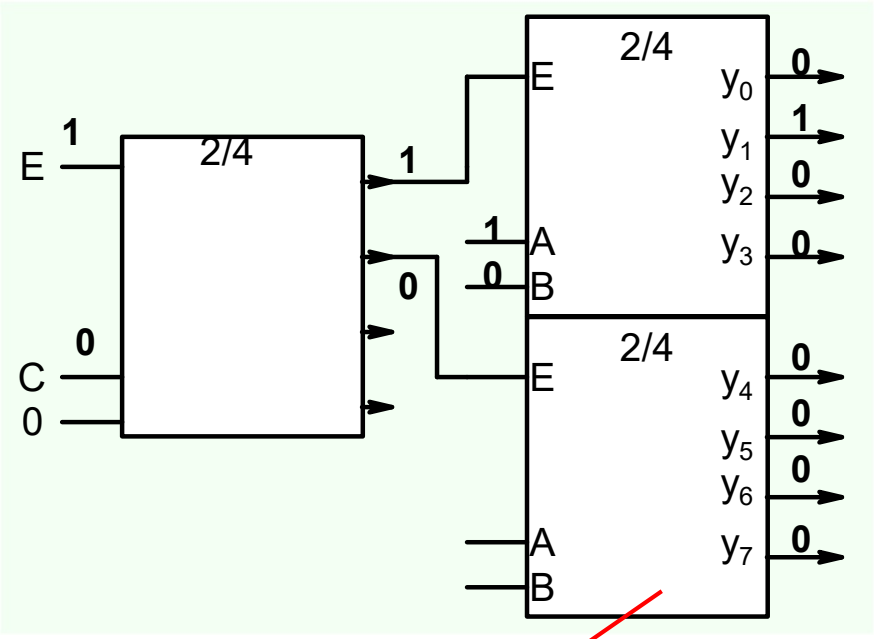
Implementing larger decoders using simpler ones.

3/8 decoder using 2/4 decoders



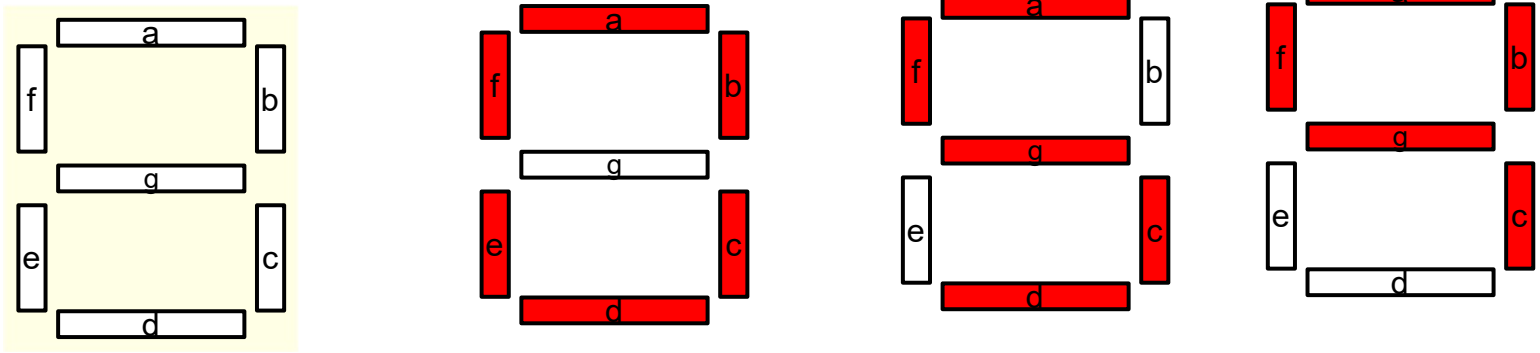
How many 2/4 decoders are required to implement a 4/16 decoder ?

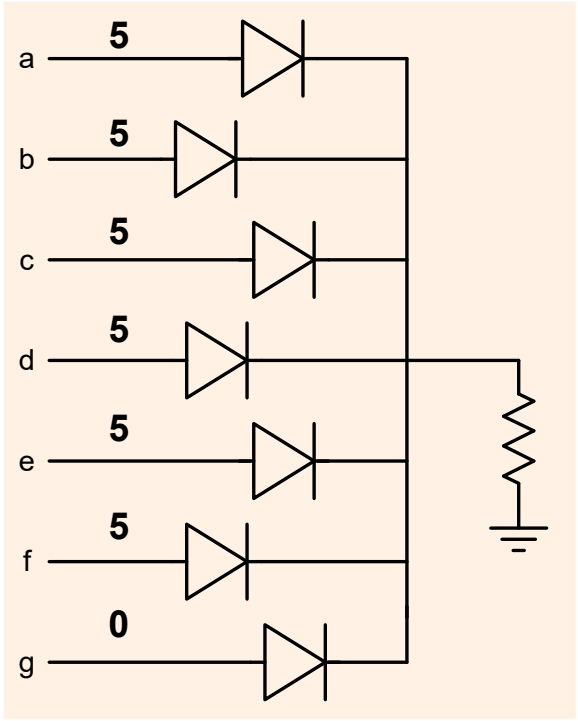
E	C	B	A	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1

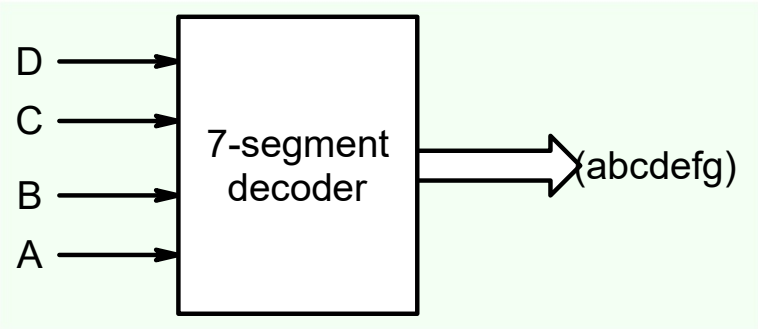


E	B	A	Y_0	Y_1	Y_2	Y_3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

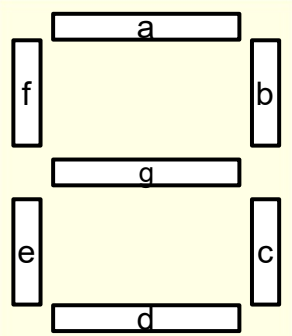
Seven segment decoder

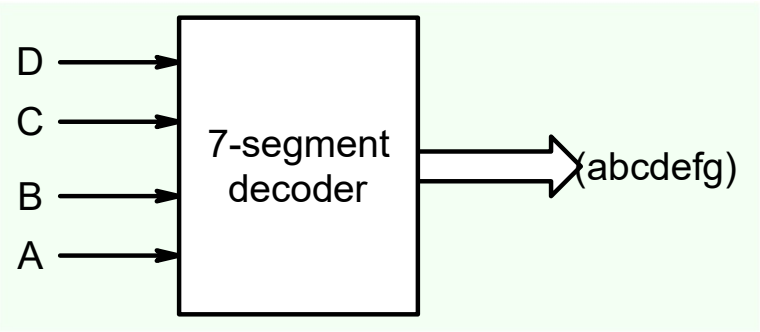






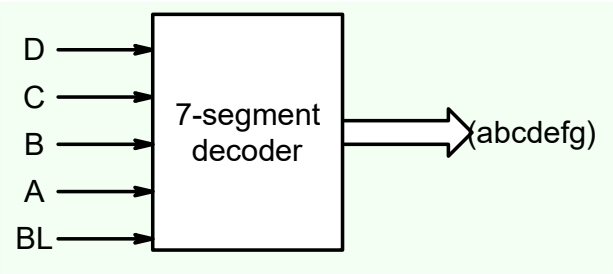
Seven segment decoder

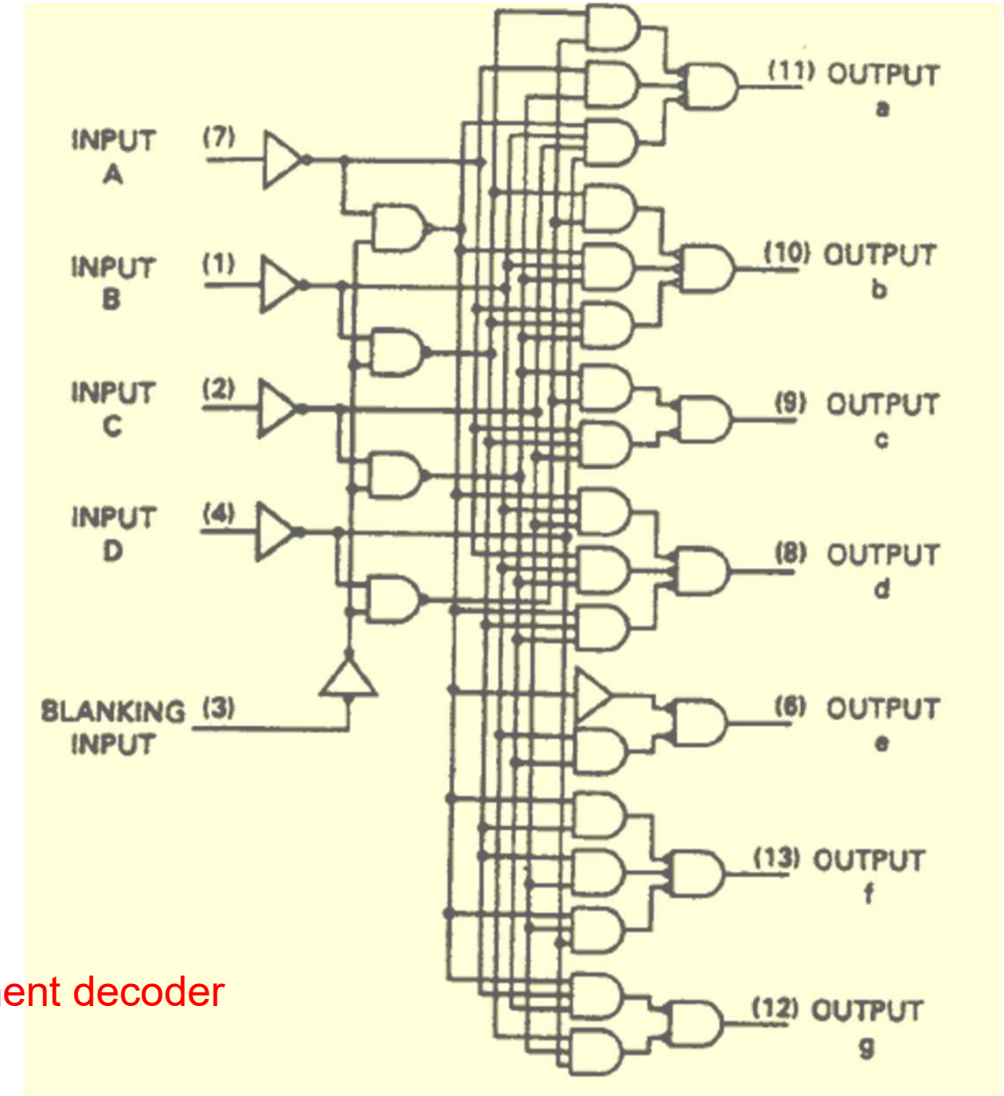




Dec or Function	Input				BI	Output						
	D	C	B	A		a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	1	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	1	0	0	1
4	0	1	0	0	1	0	1	1	0	0	1	1
5	0	1	0	1	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	0	1	1
10	1	0	1	0	1	0	0	0	1	1	0	1
11	1	0	1	1	1	0	0	1	1	0	0	1
12	1	1	0	0	1	0	1	0	0	0	1	1
13	1	1	0	1	1	1	0	0	1	0	1	1
14	1	1	1	0	1	0	0	0	1	1	1	1
15	1	1	1	1	1	0	0	0	0	0	0	0
BI	×	×	×	×	0	0	0	0	0	0	0	0

DC \ BA	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	0	0
10	1	1	0	0

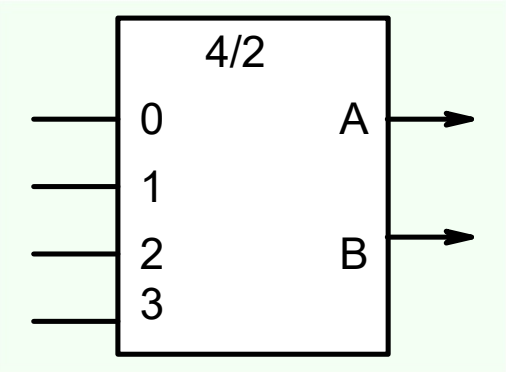




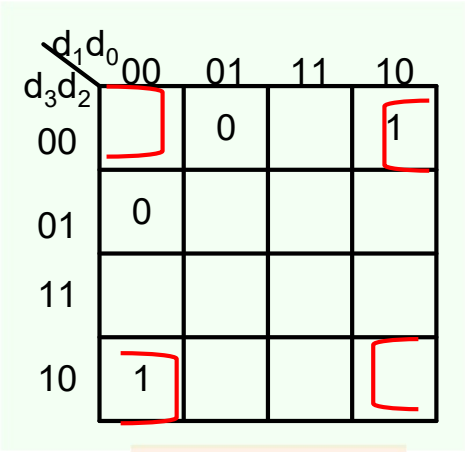
7449 BCD to seven segment decoder

Encoders

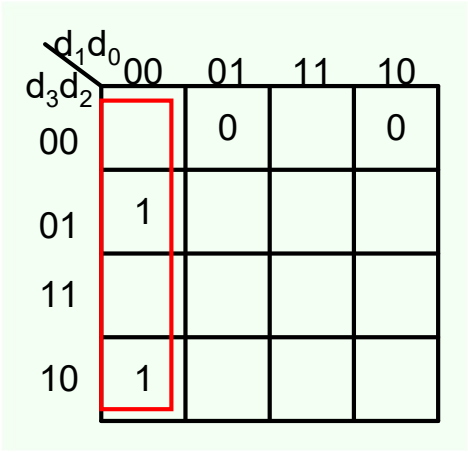
An encoder performs the inverse operation of a decoder.



d_3	d_2	d_1	d_0	B	A
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

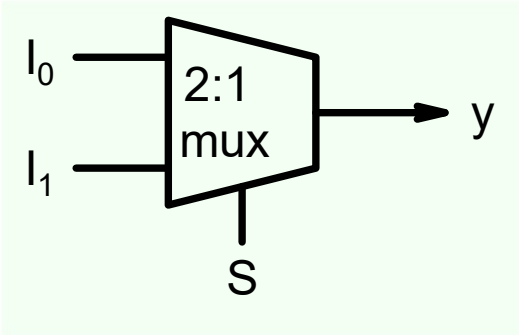


$$A = \overline{d_2}\ \overline{d_0}$$

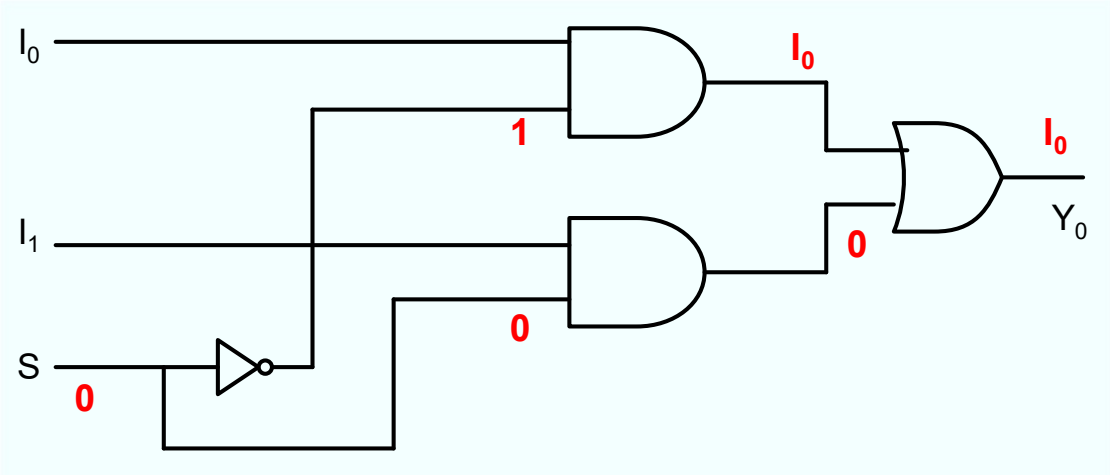


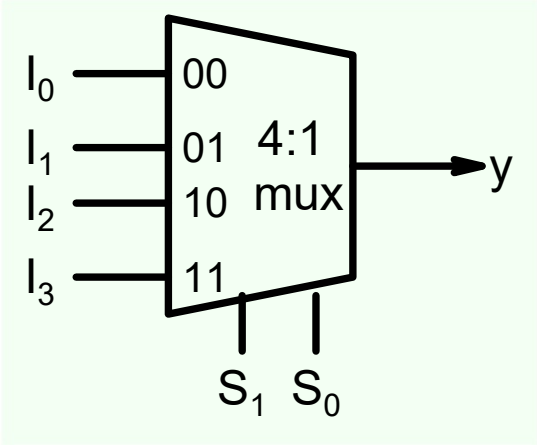
$$B = \overline{d_1}\ \overline{d_0}$$

Multiplexers

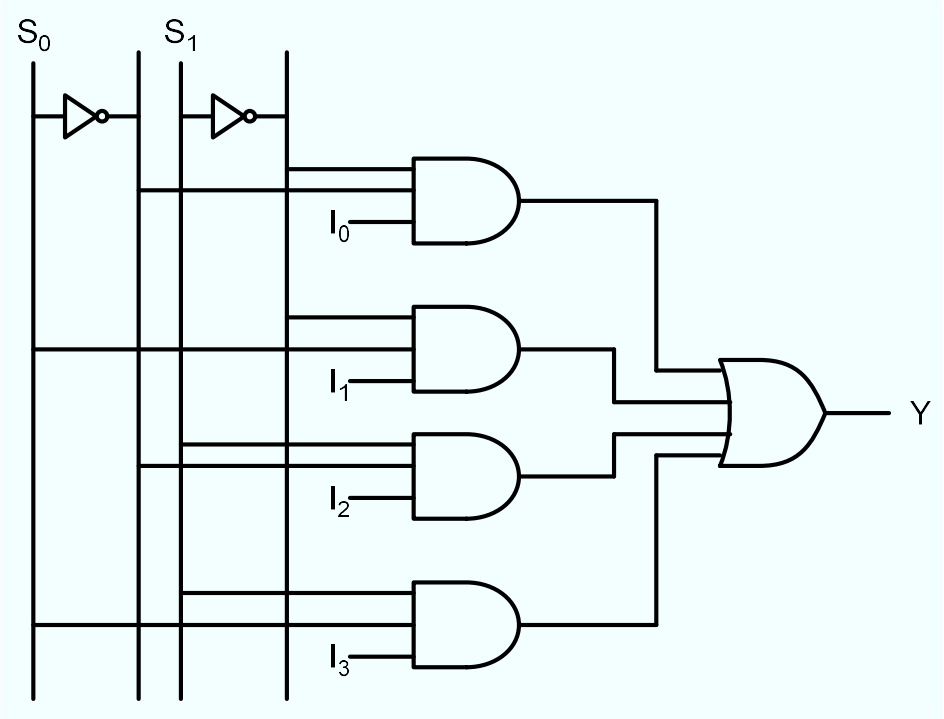


S	*y*
0	I_0
1	I_1

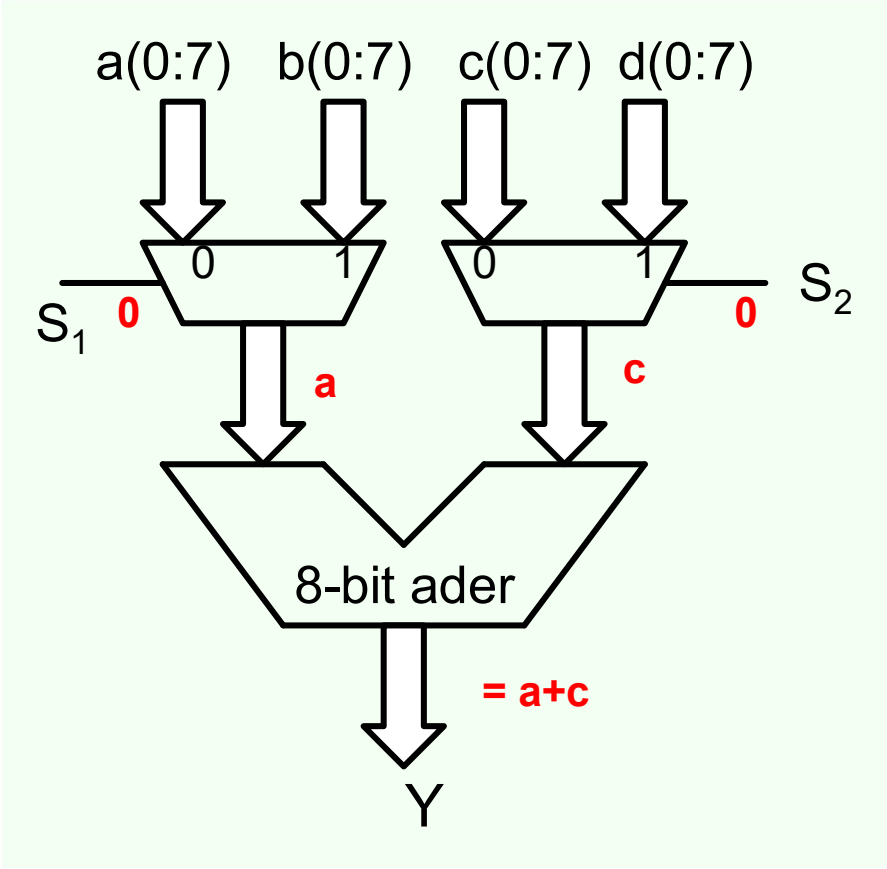




S_1	S_0	y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



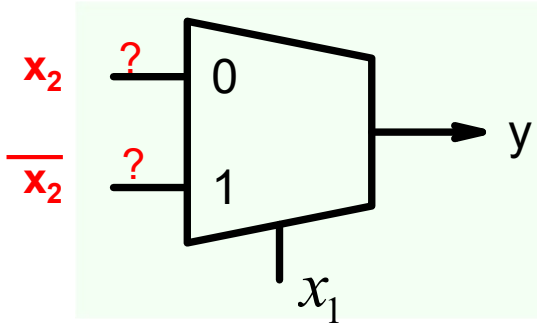
Mux is often used when resources have to be shared



S_1	S_0	y =
0	0	a+c
0	1	a+d
1	0	b+c
1	1	b+d

Implementing Boolean expressions using Multiplexers

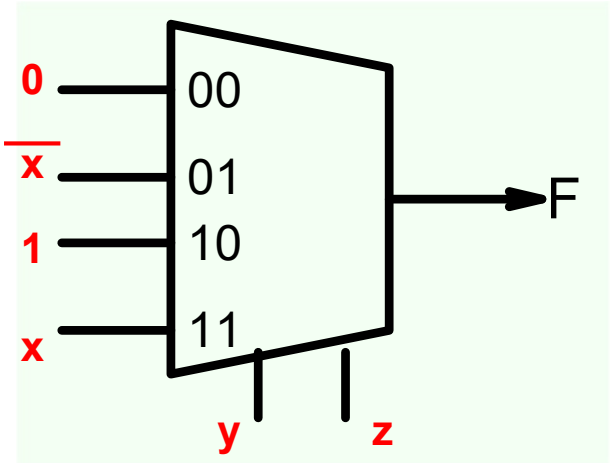
$$y = x_1\overline{x_2} + \overline{x_1}x_2$$



x_1	x_2	y	
0	0	0	
0	1	1	$y = x_2$ when $x_1 = 0$
1	0	1	
1	1	0	$y = \overline{x_2}$ when $x_1 = 1$

$$F(x, y, z) = \sum(1, 2, 6, 7)$$

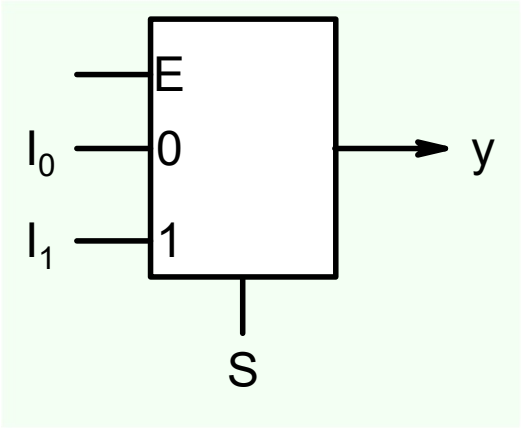
A 3 variable function can be implemented with a 4:1 mux with 2 select lines



x	y	z	F	
0	0	0	0	F = 0 when yz = 00
1	0	0	0	
0	0	1	1	F = $\overline{x}$ when yz =01
1	0	1	0	
0	1	0	1	F = 1 when yz = 10
1	1	0	1	
0	1	1	0	F = x when yz = 11
1	1	1	1	

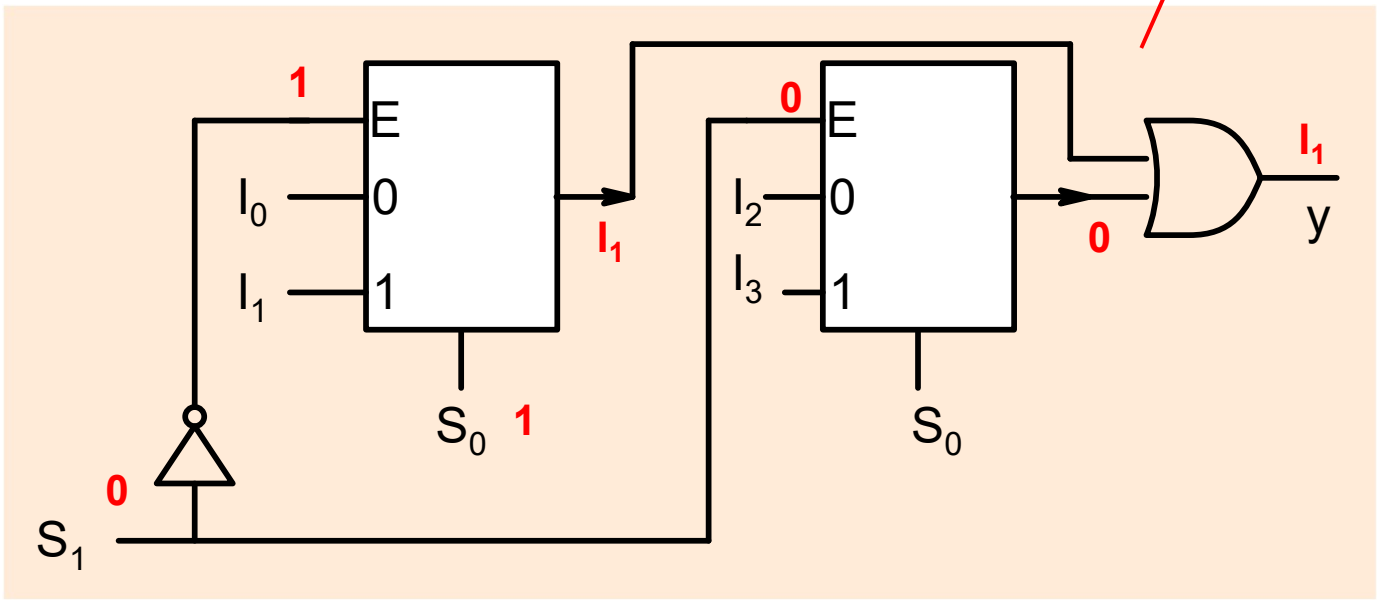
Mux is more efficient way of implementing combinational circuits as compared to decoders.

Mux. expansion

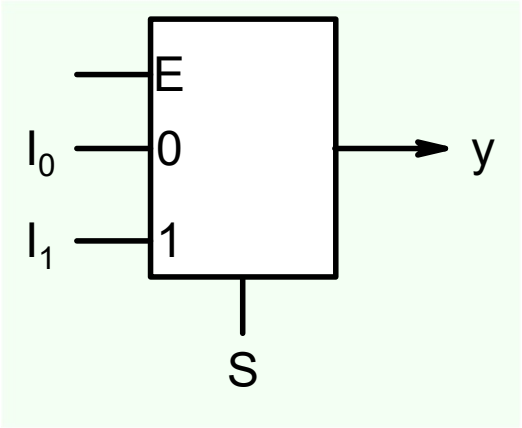


E	S	y
0	x	0
1	0	I_0
1	1	I_1

S_1	S_0	y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

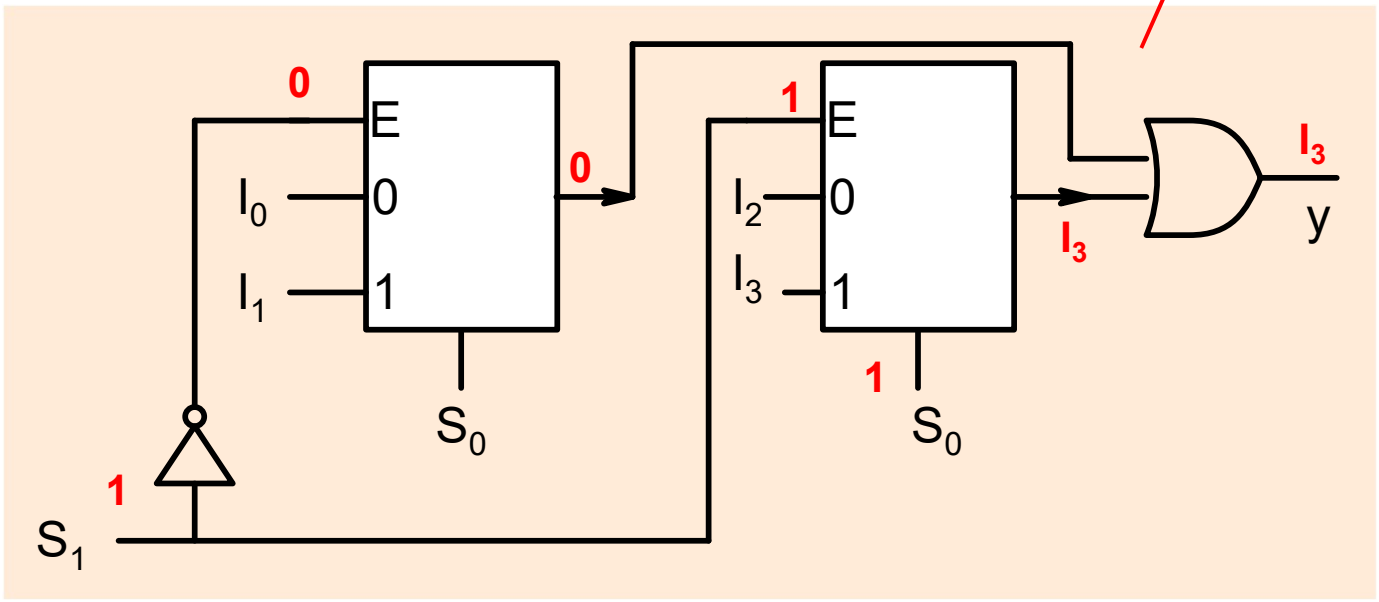


Mux. expansion

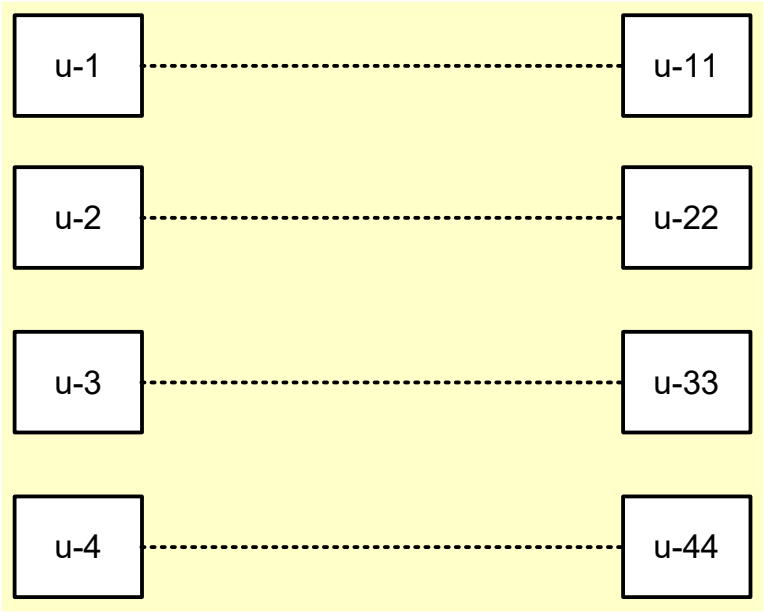


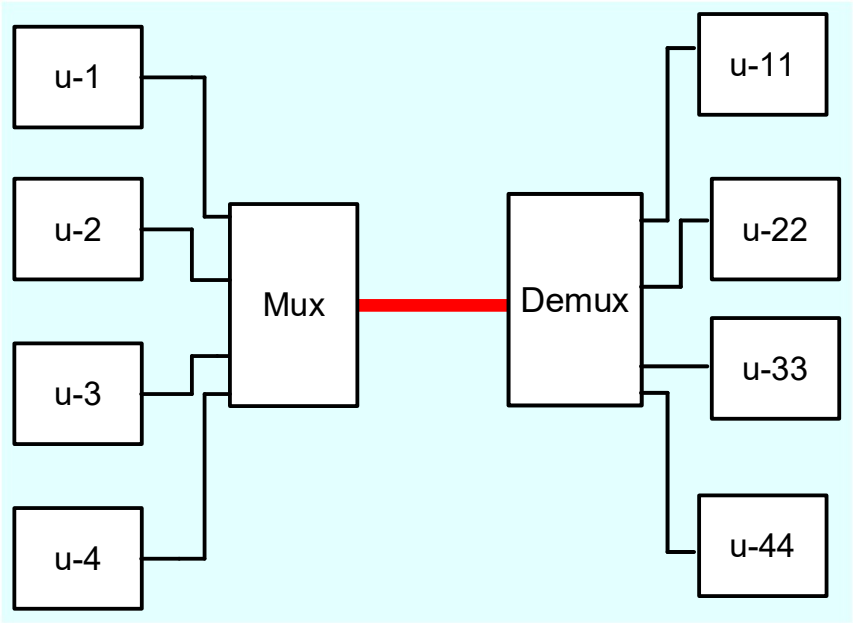
E	S	y
0	x	0
1	0	I_0
1	1	I_1

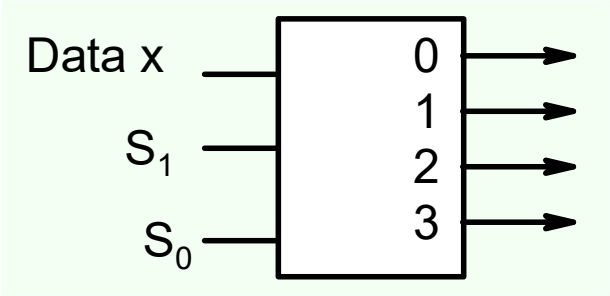
S_1	S_0	y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



DeMultiplexer

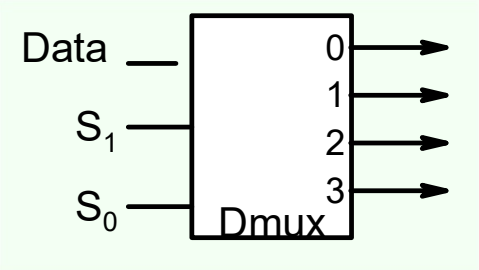


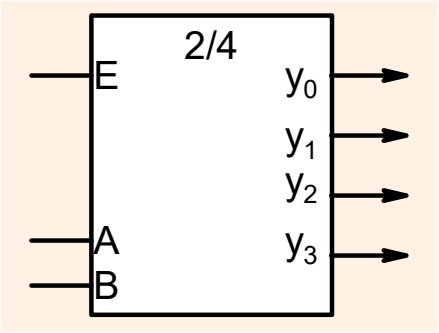




S_1	S_0	y_0	y_1	y_2	y_3
0	0	x	0	0	0
0	1	0	x	0	0
1	0	0	0	x	0
1	1	0	0	0	x

Demultiplexer is very much like a decoder





S_1	S_0	y_0	y_1	y_2	y_3
0	0	x	0	0	0
0	1	0	x	0	0
1	0	0	0	x	0
1	1	0	0	0	x

E	B	A	Y_0	Y_1	Y_2	Y_3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1